Design and Implementation of a new Inter-Process Communication Architecture for Log-based HIDS for 100 GbE Environments

---

Bachelor Thesis

by

Paul Raatschen

University of Potsdam
Institute for Computer Science
Operating Systems and Distributed Systems

Supervisors:
Prof. Dr. Bettina Schnor
M.Sc. Max Schrötter

Potsdam, April 9, 2023

Raatschen, Paul
raatschen@uni-potsdam.de
Design and Implementation of a new Inter-Process Communication Architecture for Log-based HIDS for 100 GbE Environments
Bachelor Thesis, Institute for Computer Science
University of Potsdam, April 2023

Thanks

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Potsdam, April 9, 2023

Paul Raatschen

Abstract

Deutsche Zusammenfassung

Deutsche Zusammenfassung

Contents

1 Introduction **1**

2 Background **2**
2.1 Host-based Intrusion Detection / Prevention 2
2.1.1 Fail2ban 2
2.2 Inter-Process Communication 3
2.2.1 Types of IPC 3
2.2.2 IPC based logging 3
2.3 Special Software 3
2.3.1 Hyperscan 3
2.3.2 io_uring 3
2.3.3 Trex 3

3 Design & Implementation **4**
3.1 Requirements 4
3.2 Abstract Architecture 4
3.3 Choice of IPC Type 4
3.4 Ringbuffer API 4
3.5 Proof-of-Concept IPS 4
3.6 Test Application 4

4 Evaluation **12**
4.1 Test Environment 12
4.2 Experimental Design 13
4.3 Fail2ban Replication Measurements 13
4.4 Simplefail2ban, Logfile Measurements 13
4.5 Simplefail2ban, Shared Memory Measurements 13
4.6 Shared Memory Feature Measurements 13

5 Conclusion **39**

List of Figures 40

List of Tables 41

List of Algorithms 42

A Abbreviations 43

B Source Files 44

Bibliography 45

1 Introduction

Since the advent of the Internet, bandwiths available to both commercial and private users have been ever increasing. While this opens up new possibilitys for high bandwiths network applications, it also poses new security challenges for dealing with potentially malicious network traffic. In addition to traditional firewalls, Host-based Intrusion Detection System (HIDS) are a commonly used security measure, to protect a system . Traditionally, HIDS make use of application logfiles, which are parsed for information on possible attacks and to identify malicious clients. The Intrusion Prevention System (IPS) Fail2ban[1] is is an one of the most prim

The goal of this thesis will be the design and implementation of a new Inter-Process Communication (IPC) architecture for the transmission of log messages, that is able to facilitate low latency communication between sender and receiver. Additionally, the design should be able to scale to multiple recipients, in order to accommodate more complex security system, in which several processes require access to a hosts application log. For this purpose, a Proof of Concept IPS will be developed, that utilizes the proposed IPC architecture to receive log messages and ban malicious clients in the style of Fail2ban.

This thesis will be structured as follows: The following section provides background information on relevant concepts,

2 Background

The following section introduces the concept of HIDS with the specific example of Fail2ban and presents the problem setting this thesis aims to solve. In addition to that, an overview over common types of Inter-Process Communication and existing IPC based logging solution is given. Finally, external libraries and other software used for the implementation and evaluation of the Proof-of-Concept IPS are introduced.

2.1 Host-based Intrusion Detection / Prevention

The idea of specialized software for detecting intrusion attempts and other security threads goes as far back as 1980, when James Anderson published a study on "Computer security threat monitoring and surveillance"[2]. In 1987, Dorothy Denning presented a seminal model for Intrusion Detection Systems, that suggested the use of pattern matching based on statistical analysis of audit records generated by a system, in order to detect abnormal user behavior [3]. Intrusion Detection Systems in general, gather data from a multitude of sources, which is then processed to identify and report potential threads. Host-based Introduction Detection Systems in particular, use information that is provided by the hosts under their supervision. This includes event logs of applications, as well as operating system (Operating System (OS)) based information, such as user logins, file system operations or systemcalls. The analysis of the gathered data can be divided into two categories: 1. Misuse based detection relies on predefined patterns of misuse or malicious behavior, which are then matched against the observed behavior in the data. 2. Anomaly based detection uses statistical analysis to identify deviations from the norm, thereby also being able identify attacks, that have not been previously observed [4]. Intrusion Prevention Systems (IPS) constitute a special class of IDS, which are not only capable of detecting an attack, but also take measure to prevent or mitigate it.

2.1.1 Fail2ban

Fail2ban is a open source IPS for POSIX Systems, that is widely used to protect web servers, for instance against brute-force login attempts, as well as other types of attacks []. To identify potentially malicious clients, Fail2ban makes use of application logs, that are parsed based on a predefined filter. Fail2ban uses configuration units called 'Jails' , that allow for the customization to a wide range of applications. A Jail defines the path to the application log, the filter being applied to the log messages within the logfile and an action, that is executed on client matching the filter criteria. In addition to that, Jails contain further parameters, such as the threshold of matches a client need to reach, in order for the action to be executed, as well as the duration of the action. The filter component of a Jail defines a set of regular expressions, that are used to identify certain events in a log, like an unsuccessful login attempts or the exceeding of a rate

limit. the filter also obtains a clients IP address as well as the date and time of the log messages, to determine, if the event occurred in a relevant time frame.

2.2 Inter-Process Communication

2.2.1 Types of IPC

2.2.2 IPC based logging

Syslog, Rsyslog

2.3 Special Software

2.3.1 Hyperscan

Hyperscan is a open source regular expressions matching engine developed by Intel. It is specifically designed for high performance use cases, such as the application in security contexts and is being used by the intrusion detection systems Snort and Suricata. The process of regular expressions matching with Hyperscan is separated into compile- and run-time. At compile-time a set regular expressions in string representation are compiled into a database, with additional configuration options

2.3.2 io_uring

2.3.3 Trex

3 Design & Implementation

The following section introduces the design and implementation of the proposed IPC architecture and the proof of concept IPS, as well as auxiliary libraries and applications. First, the requirements and the overall purpose of the design are defined and translated into an abstract architecture. Subsequently,

3.1 Requirements

3.2 Abstract Architecture

3.3 Choice of IPC Type

3.4 Ringbuffer API

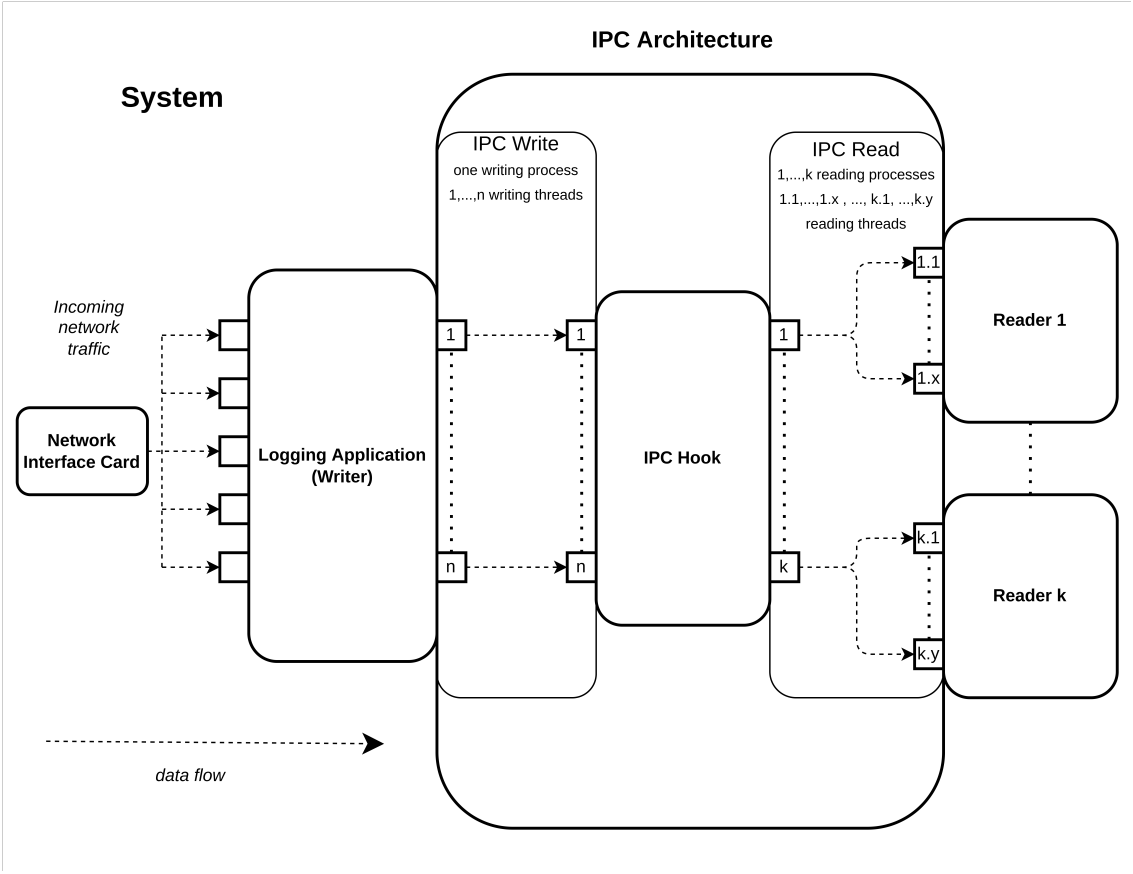
```
1  struct shmrbuf_writer_arg_t
2  {
3      const char * shm_key;
4      uint16_t line_size;
5      uint32_t line_count;
6      uint8_t segment_count, reader_count;
7      struct shmrbuf_global_hdr_t * global_hdr;
8      struct shmrbuf_seg_whdr_t * segment_hdrs;
9      int flags, shm_id;
10 };
```

Algorithm 3.1: this is some text

3.5 Proof-of-Concept IPS

3.6 Test Application

Figure 3.1: Abstract IPC architecture



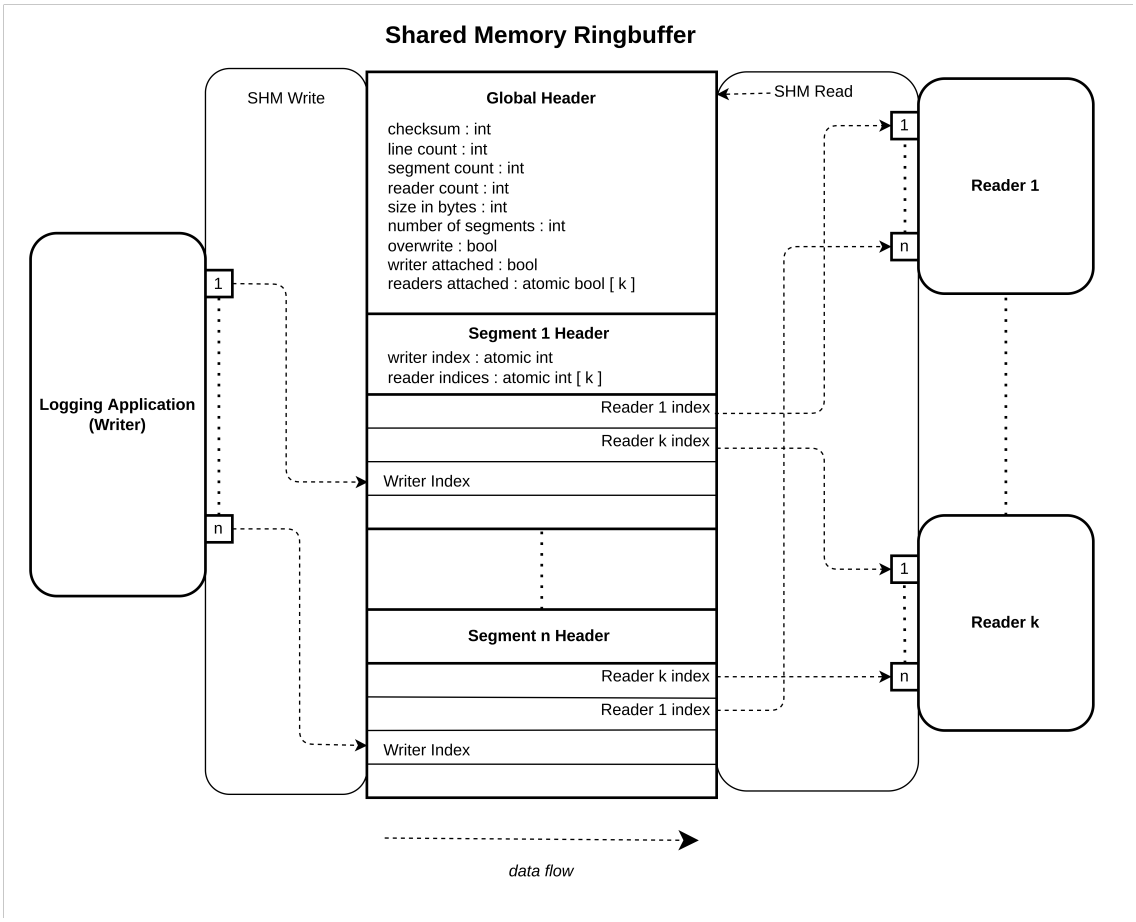


Figure 3.2: Architecture for the single-writer multi-reader shared memory ringbuffer for the transmission of log messages.

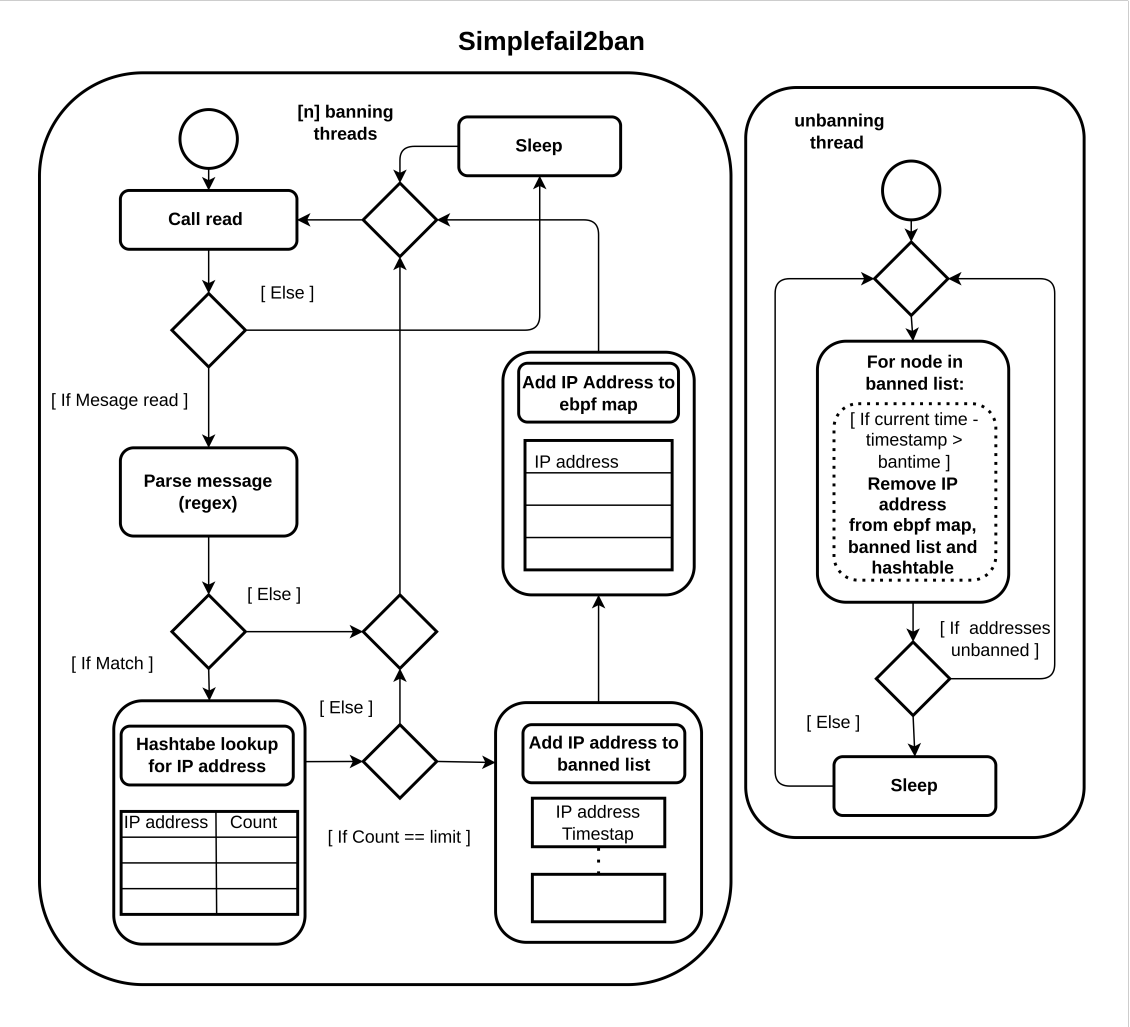


Figure 3.3: Activity diagram for the proof-of-concept IPS implementation. A variable number of "banning threads" receive log messages from a host and parse them with a predefined regular expressions. For messages that match the expression, the clients IP address is extracted from the log message and added to a hashtable, that keeps count of the number of matches per address. If the count reaches the configured limit, the address is added to the list of banned addresses with a current timestamp and inserted into the eBPF map. One "unbanning thread" routinely iterates through the banned list and checks, if a clients bantime hast elapsed. Clients with an elapsed bantime are removed from the eBPF map, banned list and hashtable.

```
1  // Creates the ringbuffer or attaches to an existing one
2  int shmrbuf_init(union shmrbuf_arg_t * args,
3                   enum shmrbuf_role_t role);
4
5  // Detaches from the ringbuffer and removes the  memory
       segment, if no other process is attached
6  int shmrbuf_finalize(union shmrbuf_arg_t *,
7                       enum shmrbuf_role_t role);
```

Algorithm 3.2: this is some text

```
1  // Writes a single line to a segment
2  int shmrbuf_write(struct shmrbuf_writer_arg_t * args,
3                    void * src, uint16_t wsize,
4                    uint8_t segment_id);
5
6  // Writes multiple lines to a segment
7  int shmrbuf_writev(struct shmrbuf_writer_arg_t * args,
8                     struct iovec * iovecs,
9                     uint16_t vsize,
10                    uint8_t segment_id);
```

Algorithm 3.3: this is some text

Number of Insertions	Collisions IPv4 [%]	Collisions IPv6 [%]	Collisions IPv4 & IPv6 [%]
65534	5.33	5.29	5.28
131068	10.17	10.17	10.15
600011	36.81	36.77	36.8

Table 3.1: Some text

Function	Execution Time IPv4 [Second]	Execution Time IPv4 [Second]
inet_ntop	1.29	3.98
ip_to_str	0.21	0.53

Table 3.2: Some text

```
1  // Reads a single line from a segment
2  int shmrbuf_read(struct shmrbuf_reader_arg_t * args,
3                   void * rbuf,
4                   uint16_t bufsize,
5                   uint8_t segment_id);
6
7  // Reads multiple lines from a segment
8  int shmrbuf_readv(struct shmrbuf_reader_arg_t * args,
9                    struct iovec * iovecs,
10                   uint16_t vsize,
11                   uint16_t bufsize,
12                   uint8_t segment_id);
13
14 // Reads a line from a segment ouf of a specified range.
15 int shmrbuf_read_rng(struct shmrbuf_reader_arg_t * args,
16                      void * rbuf,
17                      uint16_t bufsize,
18                      uint8_t lower,
19                      uint8_t upper,
20                      bool * wsteal);
21
22 // Reads multiple lines from a range of segments
23 int shmrbuf_readv_rng(struct shmrbuf_reader_arg_t * args,
24                       struct iovec * iovecs,
25                       uint16_t vsize,
26                       uint16_t bufsize,
27                       uint8_t lower,
28                       uint8_t upper,
29                       uint16_t * wsteal);
```

Algorithm 3.4: this is some text

```
1  struct shmrbuf_reader_arg_t
2  {
3      const char * shm_key;
4      int shm_id, flags;
5      uint8_t reader_id;
6      struct shmrbuf_global_hdr_t * global_hdr;
7      struct shmrbuf_seg_rhdr_t * segment_hdrs;
8  };
```

Algorithm 3.5: this is some text

```
struct shmrbuf_global_hdr_t
{
    uint32_t checksum;
    uint8_t segment_count, reader_count;
    uint16_t line_size;
    uint32_t line_count;
    bool overwrite;
    atomic_bool writer_att , first_reader_att;
};
```

Algorithm 3.6: this is some text

```
struct shmrbuf_seg_rhdr_t
{
    atomic_uint_fast32_t * write_index, * read_index;
    pthread_mutex_t segment_lock;
    void * data;
};
```

Algorithm 3.7: this is some text

```
struct shmrbuf_seg_whdr_t
{
    atomic_uint_fast32_t * write_index, * first_reader;
    void * data;
};
```

Algorithm 3.8: this is some text

```
// Reads a single line from a file (buffered)
int uring_getline(struct file_io_t * fio_arg,
                  char ** lineptr);

// Reads multiple lines from a file (buffered)
int uring_getlines(struct file_io_t * fio_arg,
                   struct iovec * ivoecs,
                   uint16_t vsize,
                   uint16_t bufsize);
```

Algorithm 3.9: this is some text

```
1  // Struct to store a single hashtable entry
2  struct ip_hashbin_t
3  {
4     void * key;
5     int domain;
6     uint32_t count;
7     pthread_mutex_t lock;
8     struct ip_hashbin_t * next;
9
10 };
```

Algorithm 3.10: this is some text

4 Evaluation

The following section presents the performance evaluation of the proof of concept IPS Simplefail2ban, in conjunction with the test application. First, an overview over the test environment and the experimental design is given and the Fail2Ban performance issues discovered by Florian Mikolajczak are replicated for the test application. Subsequently, the result for Simplefail2ban in both the logfile and shared memory based variants are presented. The section concludes with an evaluation of different features of the shared memory implementation.

4.1 Test Environment

To conduct the evaluation, two machines with an identical hardware and software configuration where used. The specific hardware and software version are listed in Table 4.1. One machine served as the dut, running the test application, as well as Fail2Ban / Simplefail2ban. The second machine was used for traffic generation with trex.

Table 4.1: Hardware and Software parameters for the test environment. The measurements where conducted on two identical machines, where one machines served as the Device under Test (DUT), running the test application and Fail2Ban / Simplefail2ban, while the other generated test traffic with trex.

Hardware	
CPU	Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz
NIC	Mellanox ConnectX-6 100GbE
RAM	128GB
Software	
OS	Debian 11
Kernel	6.1.0-0
NIC Driver	mlx5_core, 6.1.0-0
Fail2Ban	0.11.2
TRex	3.02

4.2 Experimental Design

The experimental design is closely oriented towards experiment 1 conducted by Florian Mikolajczak in [5], in order to provide comparable results. In the experiment, a **DOS!** (**DOS!**) attack on a server under a normal workload is simulated, through a small stream of valid a request and a significantly larger stream of invalid request. The original experiment used a BIND9 DNS server as the target of the **DOS!** attack, which triggered log messages for clients exceeding a certain rate limit. For the following evaluation, the test application introduced in **??** will serve at the target, differentiating valid and invalid request by the first byte of the sent payload[1]. While this way of determining the validity of a request is not necessarily realistic, it is efficient and allows the test application to quickly produce log messages. The test subject of the experiment For invalid request, a log messages containing the current date, time and client IP address is written to either a logfile or the shared memory ringbuffer.

The goal of this evaluation is to test, how the proof of concept IPS implementation performs in the test scenario described above.

Following the criteria

4.3 Fail2ban Replication Measurements

4.4 Simplefail2ban, Logfile Measurements

4.5 Simplefail2ban, Shared Memory Measurements

4.6 Shared Memory Feature Measurements

[1] Request are considered invalid, if the first byte if the payload has the unsigned value 42 (ASCII Letter B)

Figure 4.1: Abstract architecture

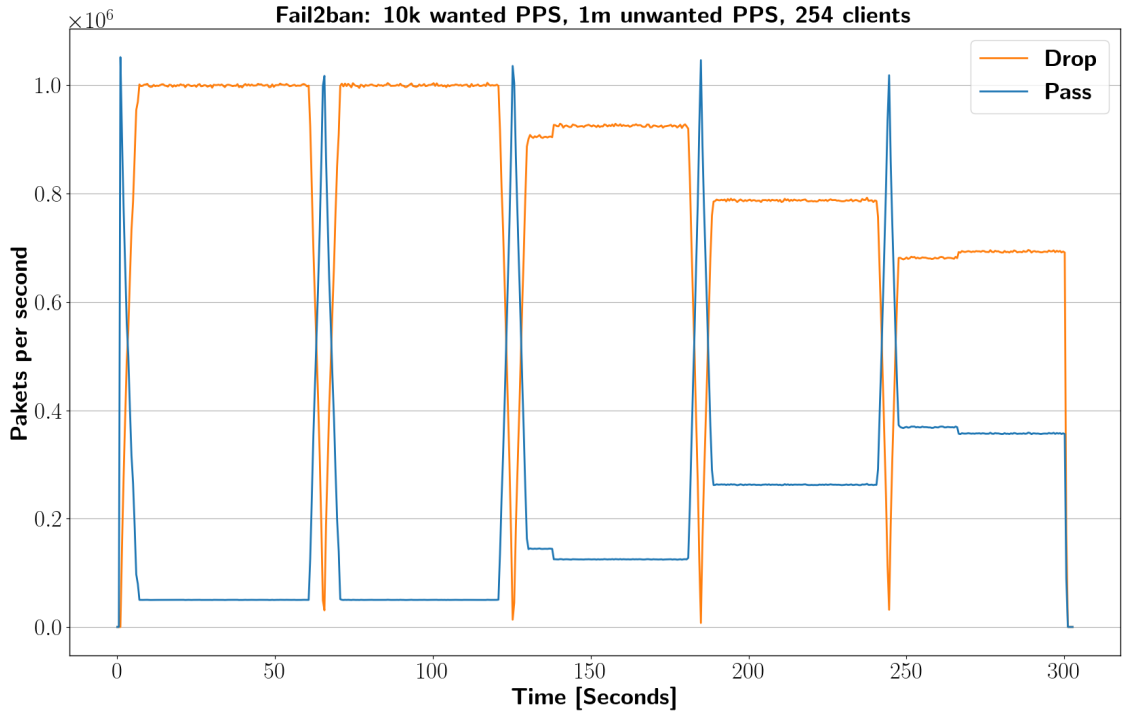
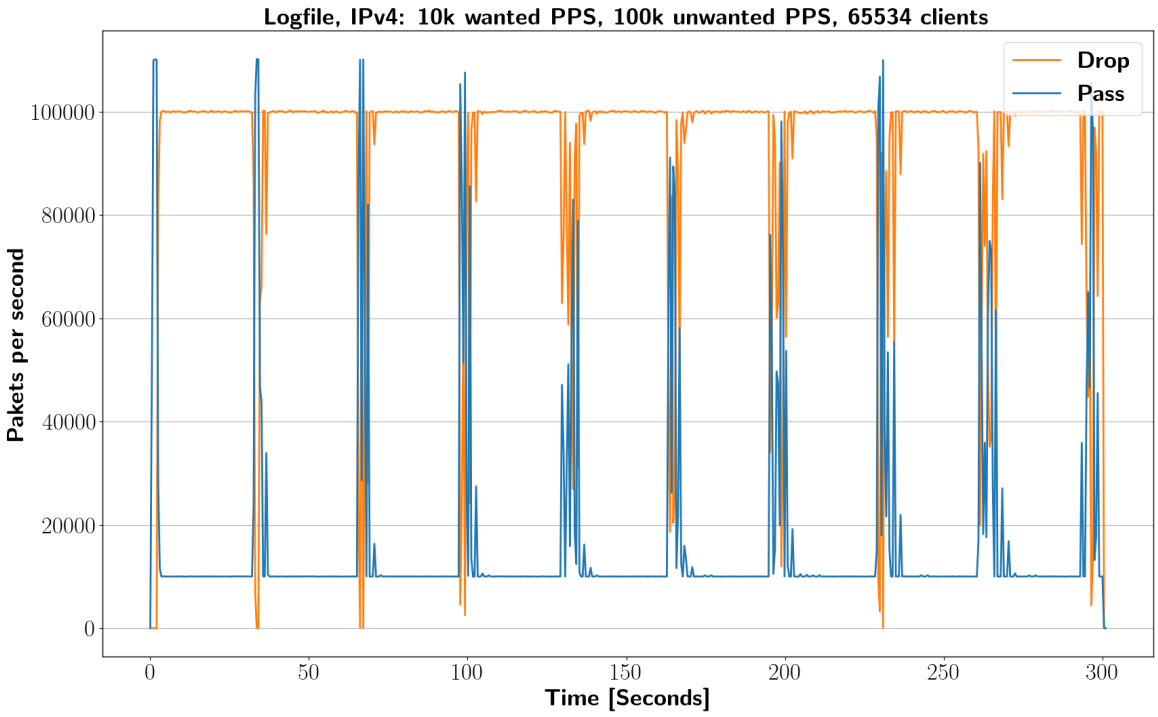
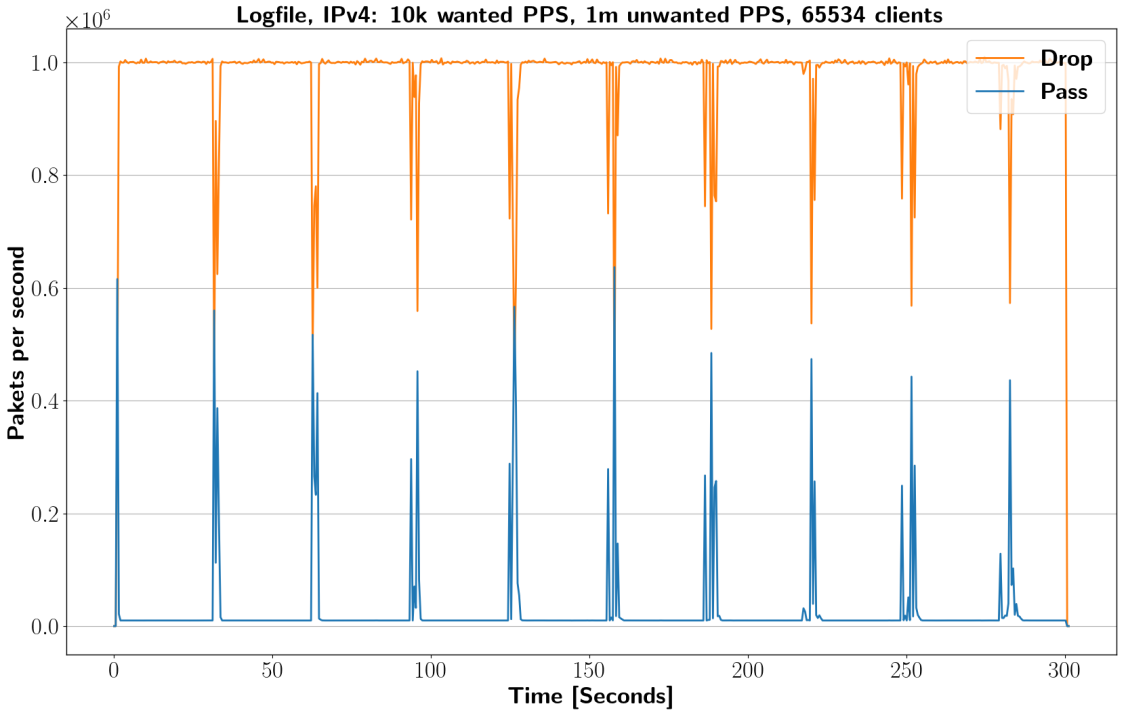


Figure 4.2: Abstract architecture



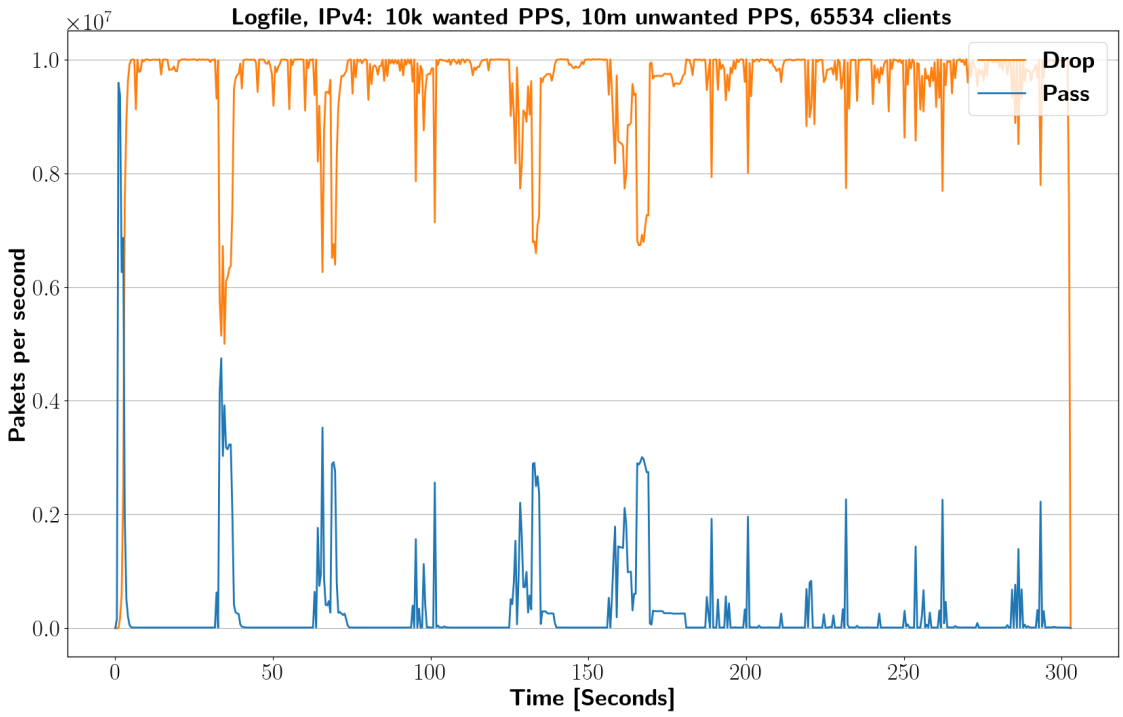
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
33	27.93	99.59	2.07	10.51

Figure 4.3: Some text



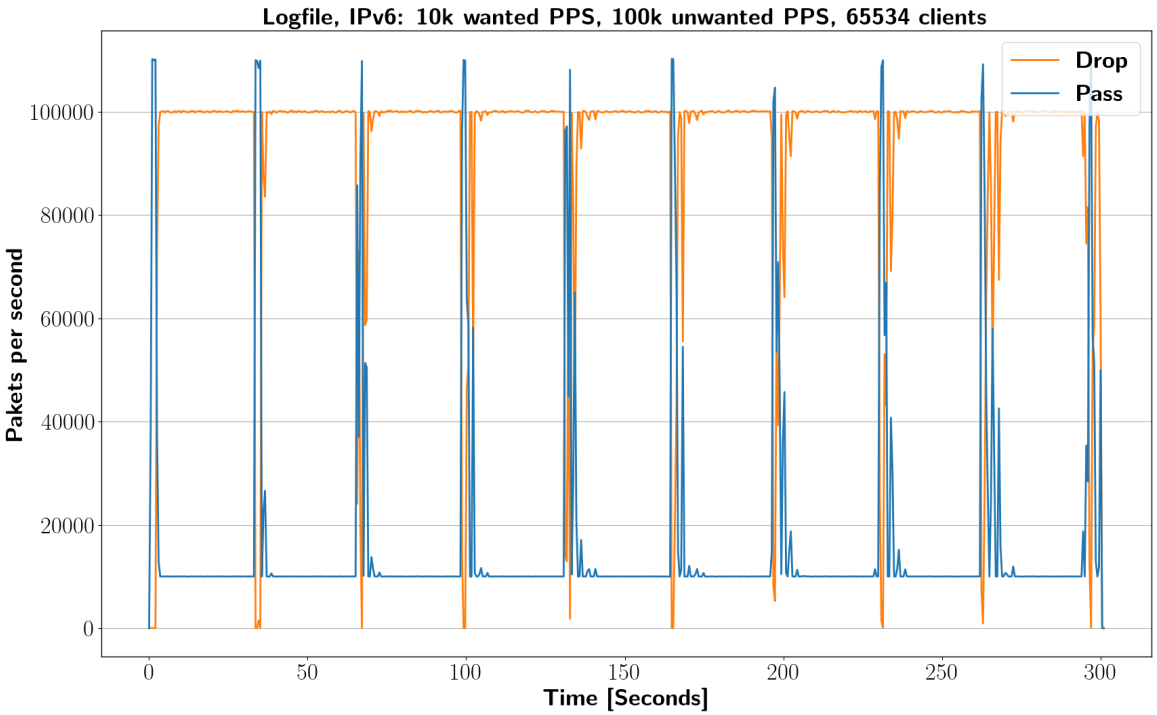
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
303	294.43	98.79	4.7	14.99

Figure 4.4: Some text



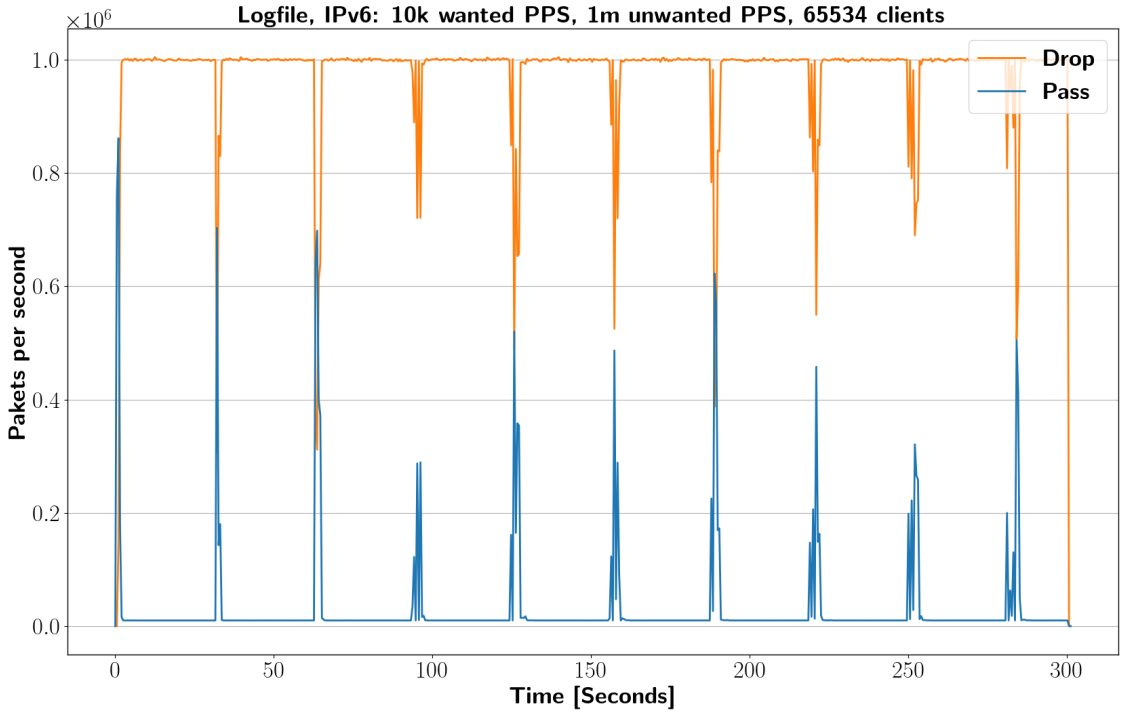
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2986.89	2884.14	96.72	34.39	83.72

Figure 4.5: Some text



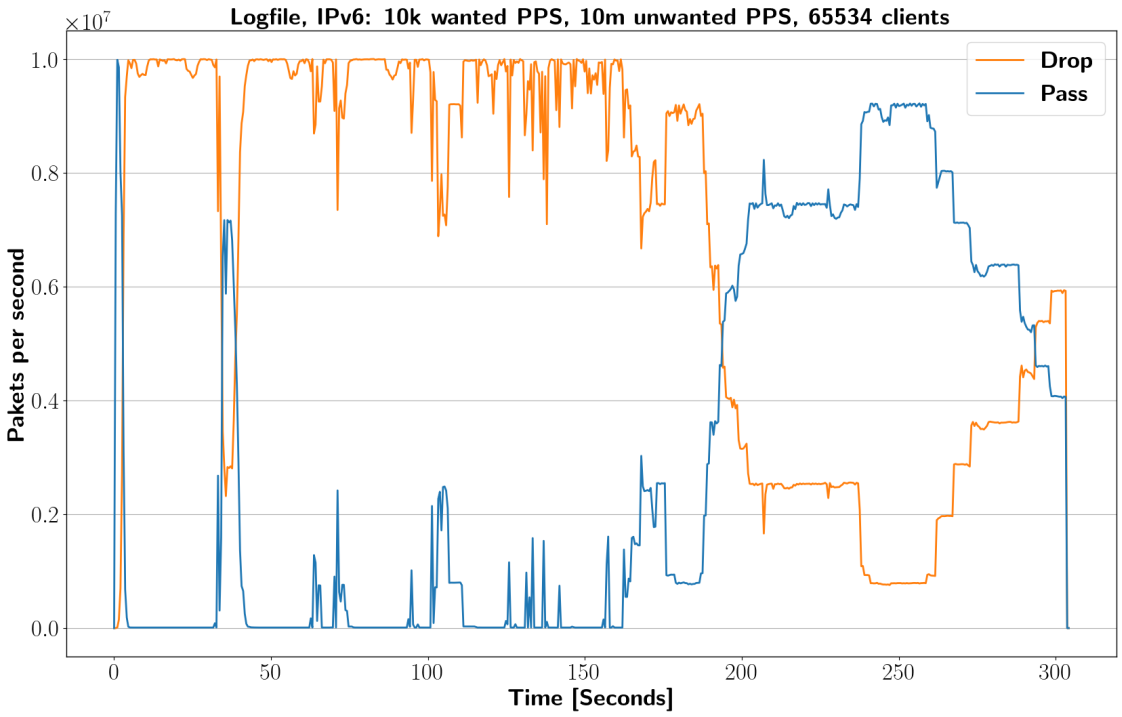
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
33	27.96	99.67	2.04	11.75

Figure 4.6: Some text



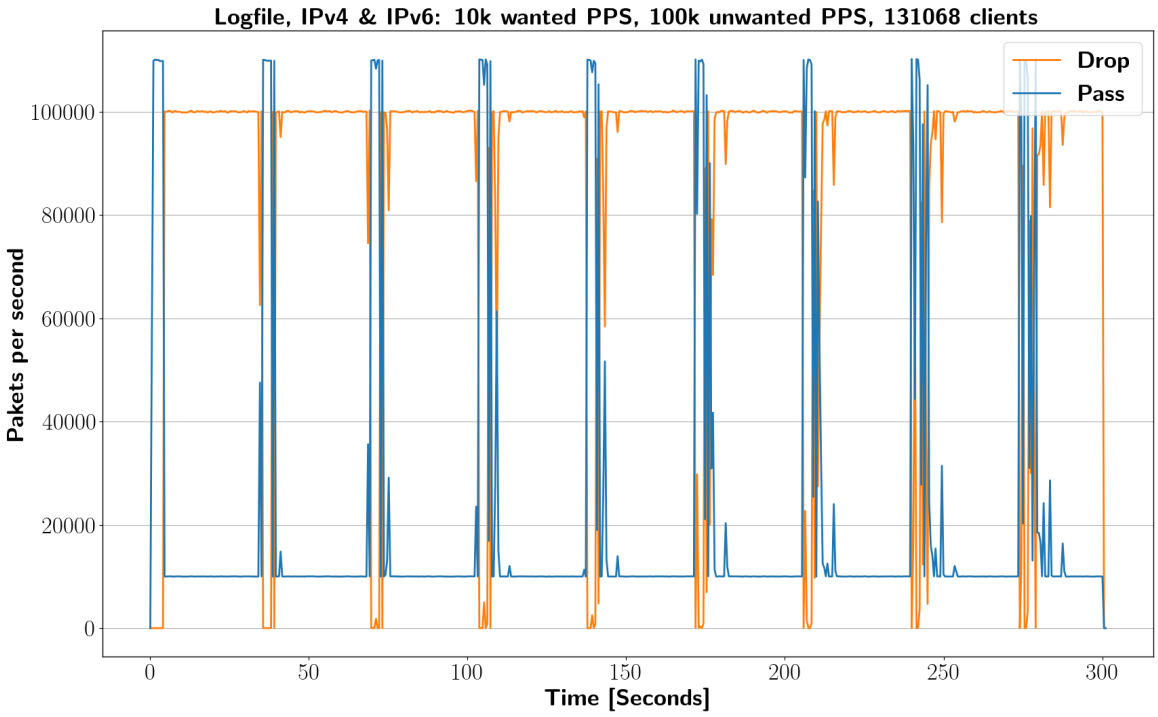
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
303	293.49	98.48	5.58	21.08

Figure 4.7: Some text



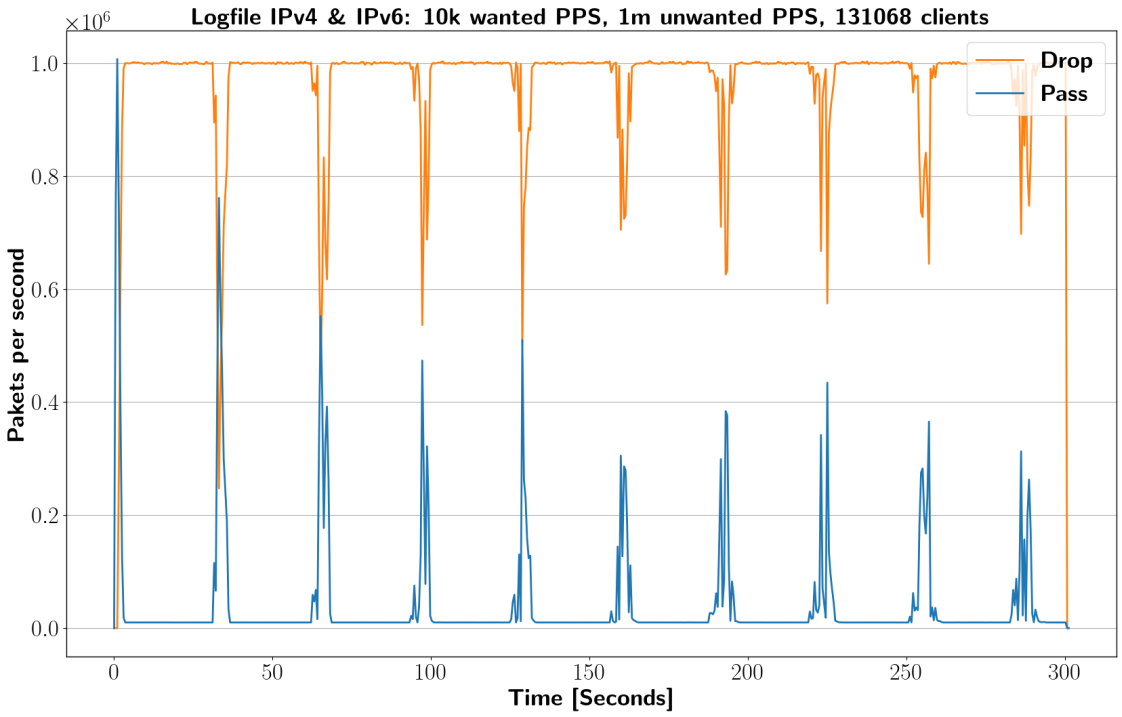
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2992.43	2065.28	69.13	205.4	346.73

Figure 4.8: Some text



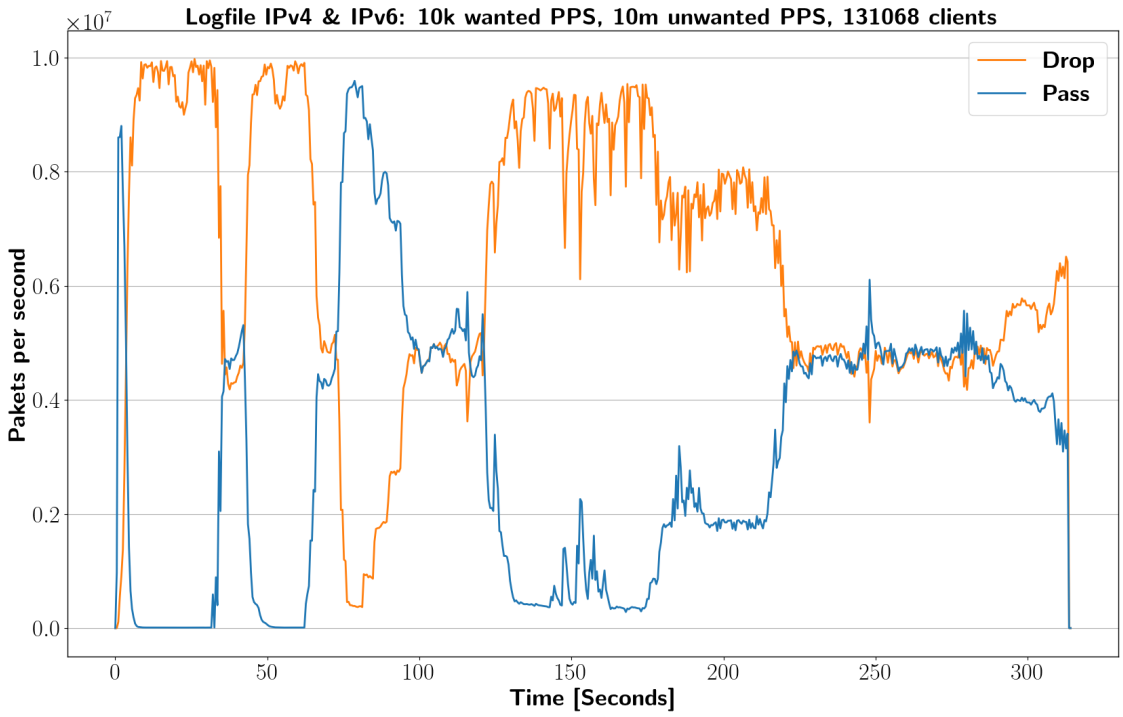
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
33	26.45	99.96	3.55	18.47

Figure 4.9: Some text



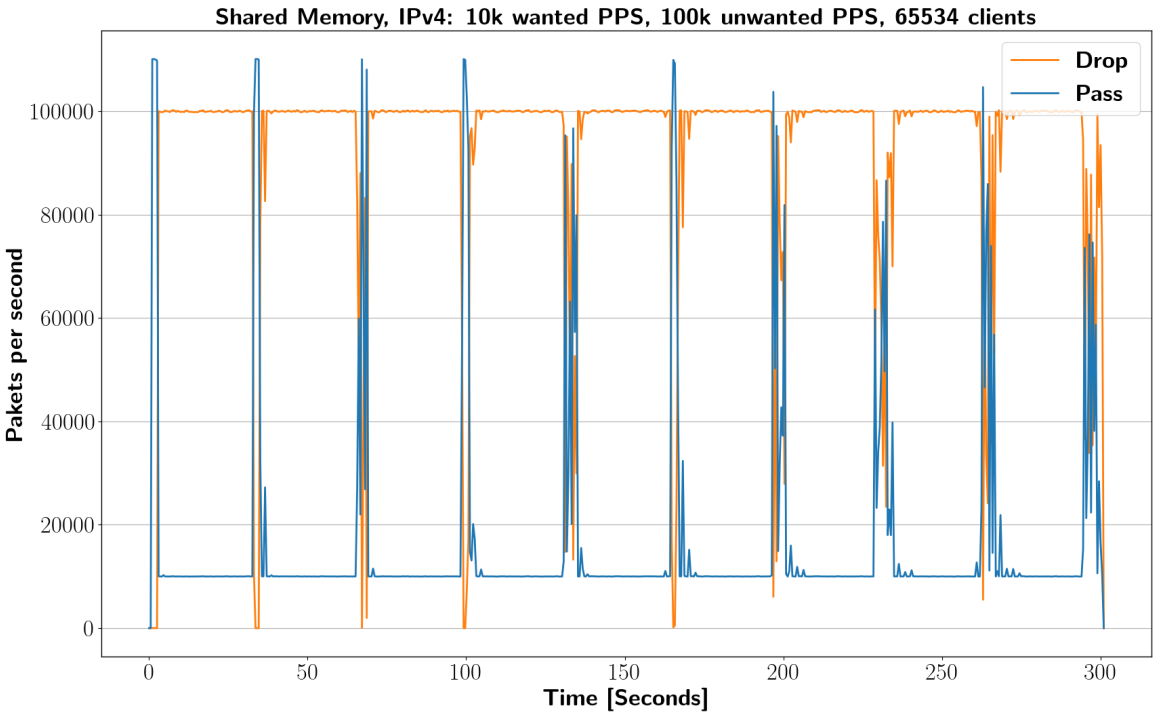
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
303	290.47	98.11	8.5	32.94

Figure 4.10: Some text



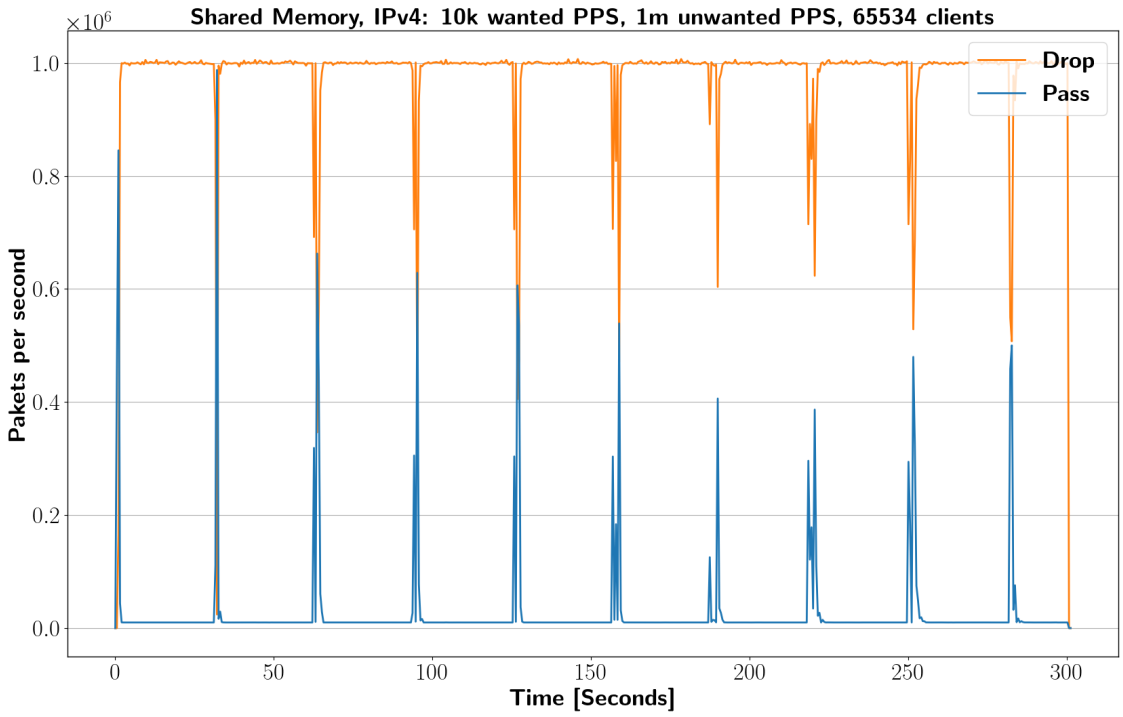
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2993.57	2014.77	67.46	213.63	382.19

Figure 4.11: Some text



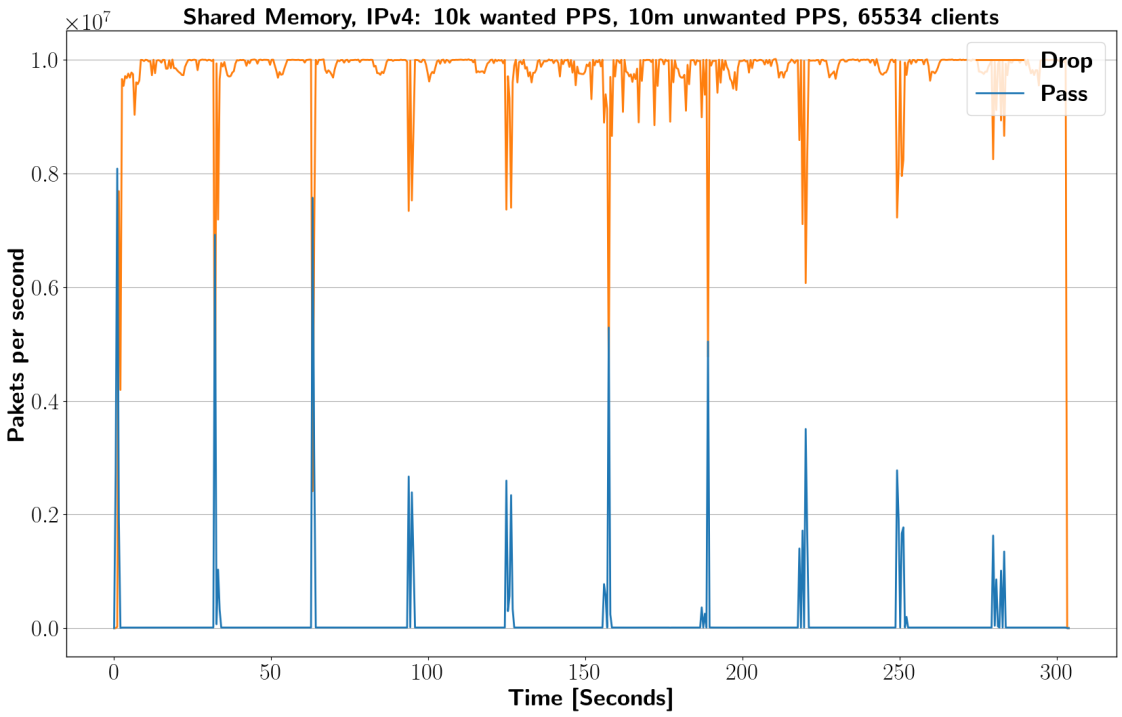
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
33	27.95	99.64	2.05	13.68

Figure 4.12: Some text



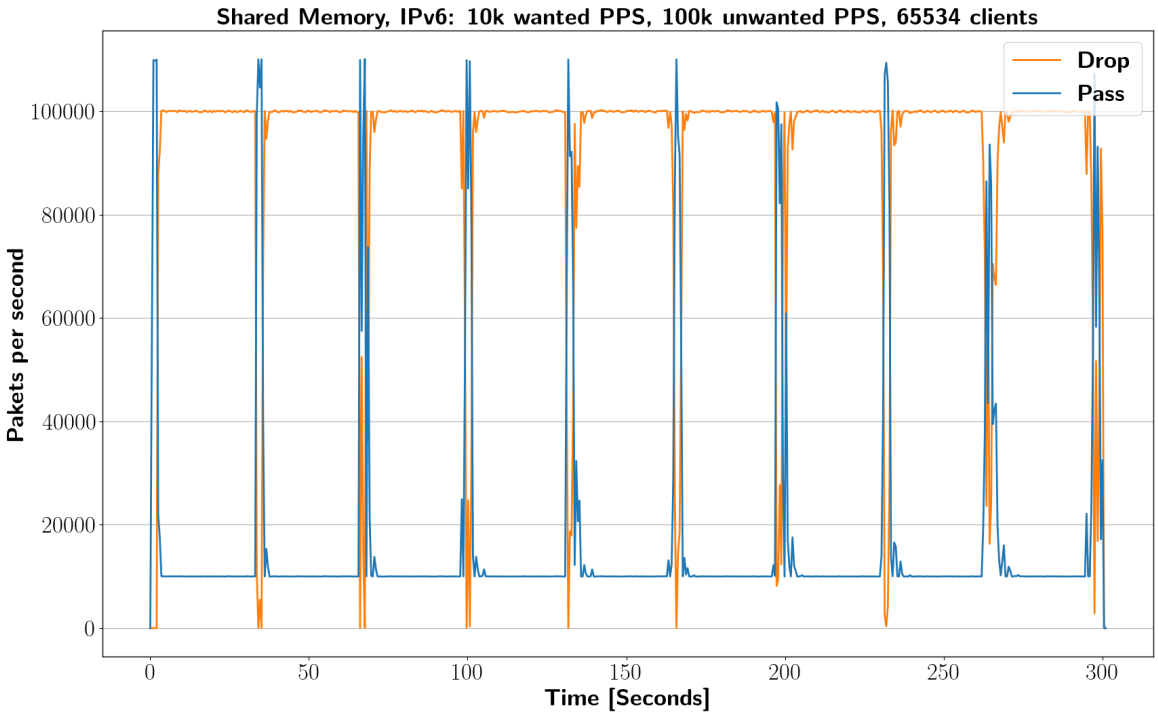
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
303	294.34	98.88	4.61	15.52

Figure 4.13: Some text



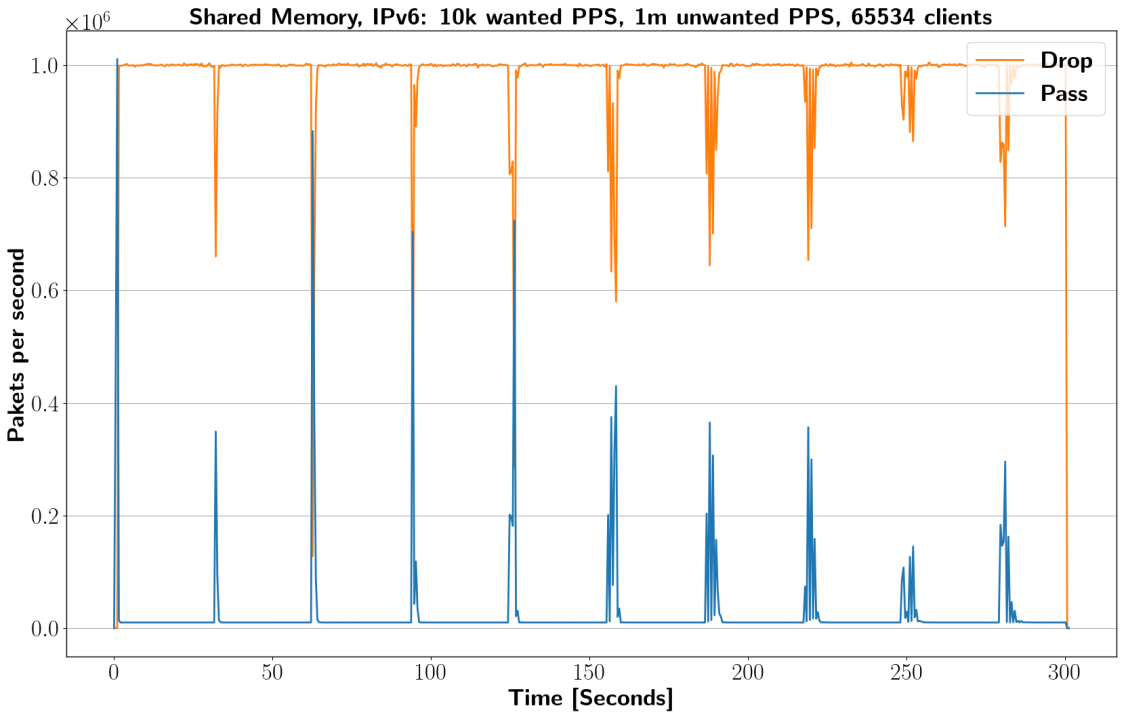
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2991.48	2948.51	96.72	7.69	25.59

Figure 4.14: Some text



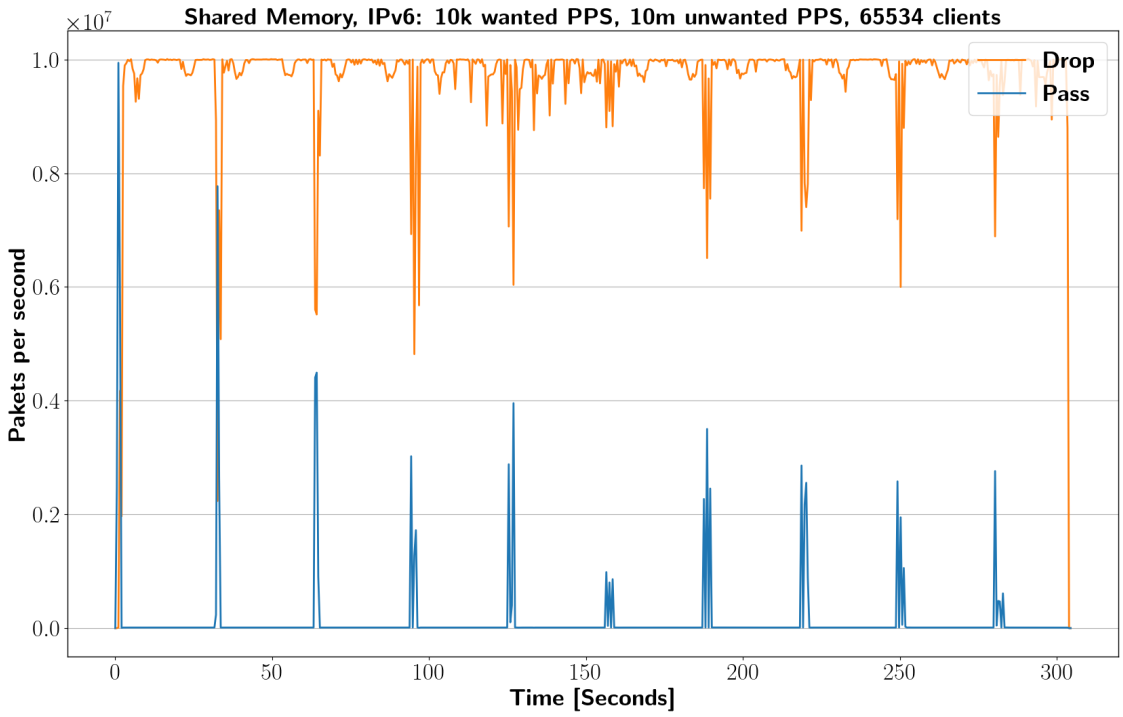
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
33	27.95	99.67	2.05	15.55

Figure 4.15: Some text



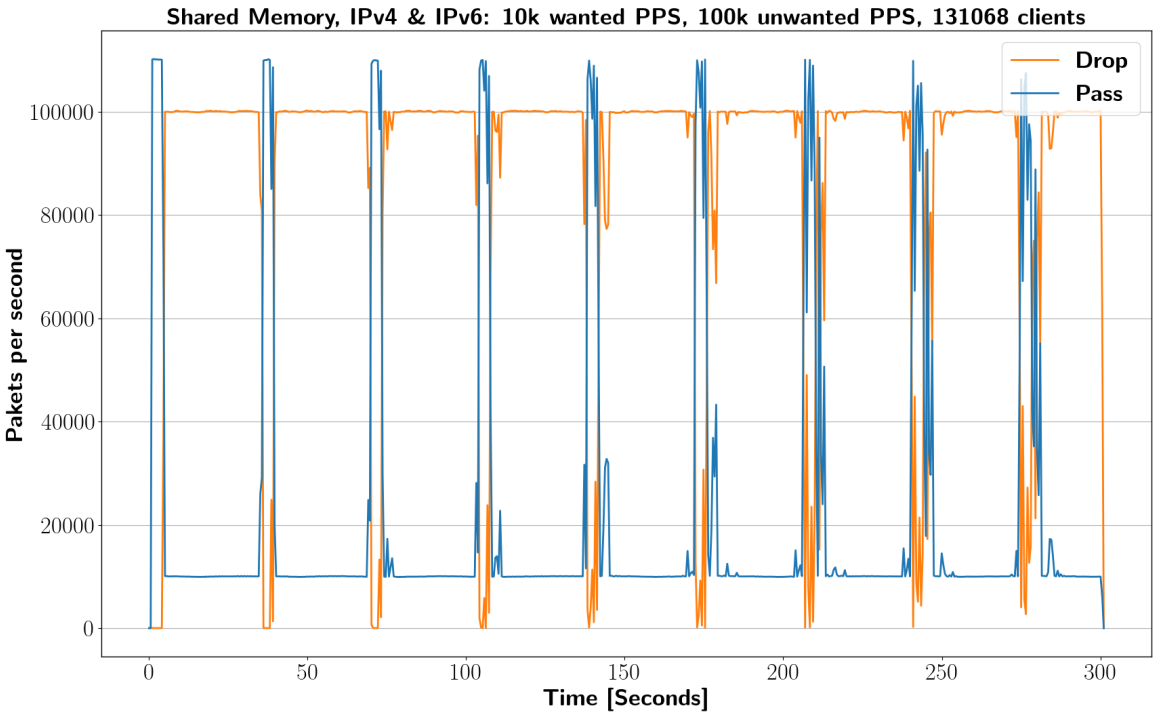
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
303	294.77	98.9	4.39	17.46

Figure 4.16: Some text



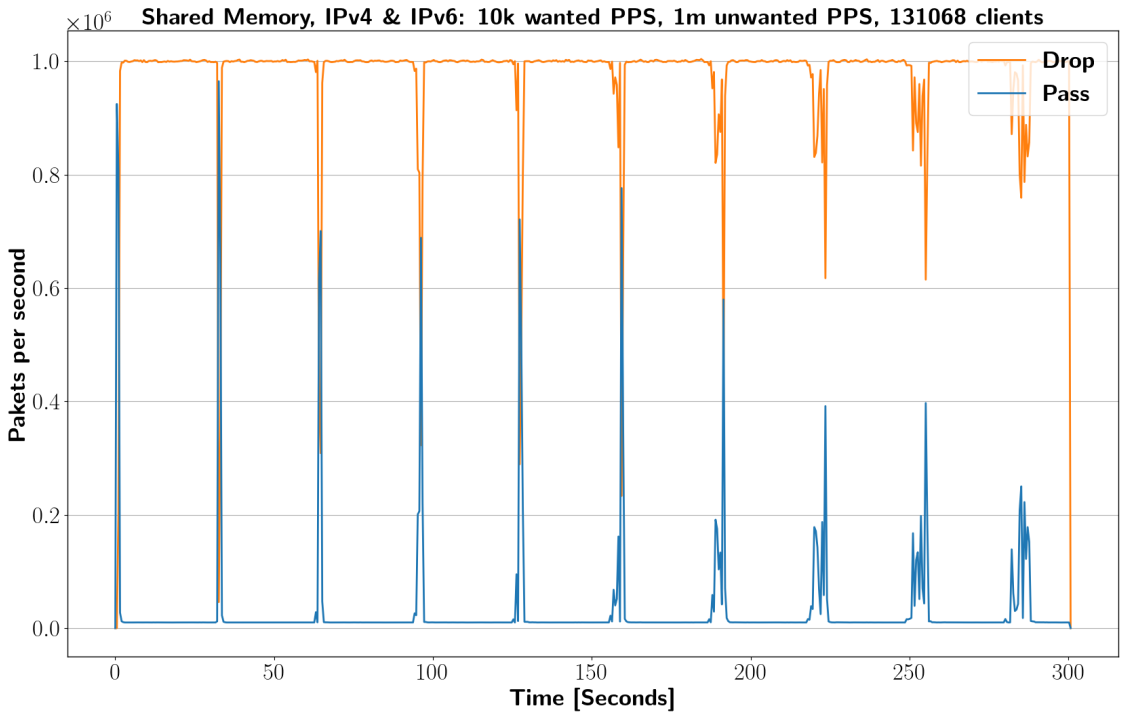
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2991.93	2947.1	98.66	8.88	32.36

Figure 4.17: Some text



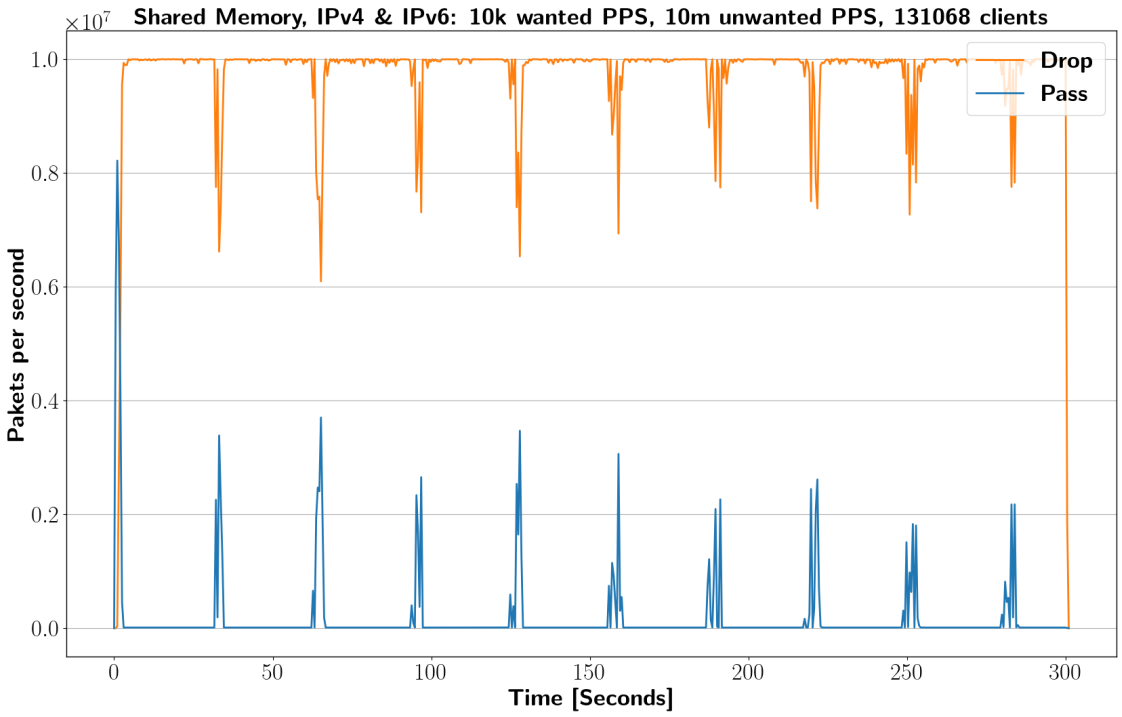
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
33	26.45	99.95	3.55	26.35

Figure 4.18: Some text



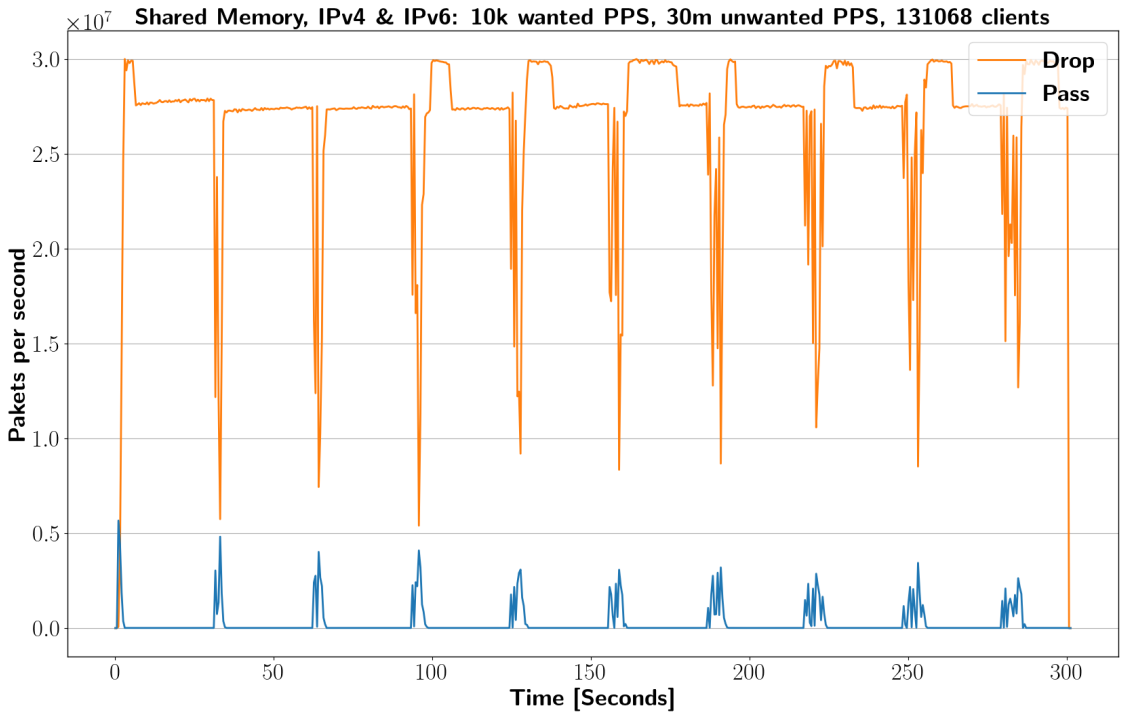
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
303	293.13	99	5.95	27.66

Figure 4.19: Some text



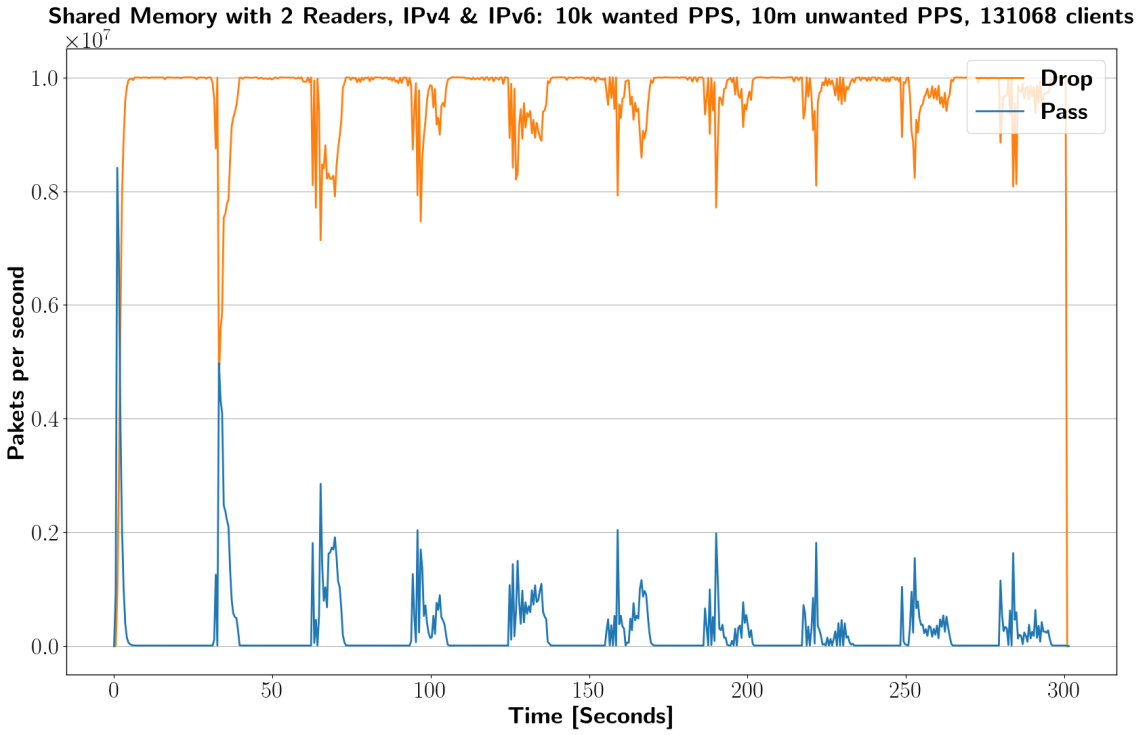
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2992.44	2939.27	98.45	16.18	65.74

Figure 4.20: Some text



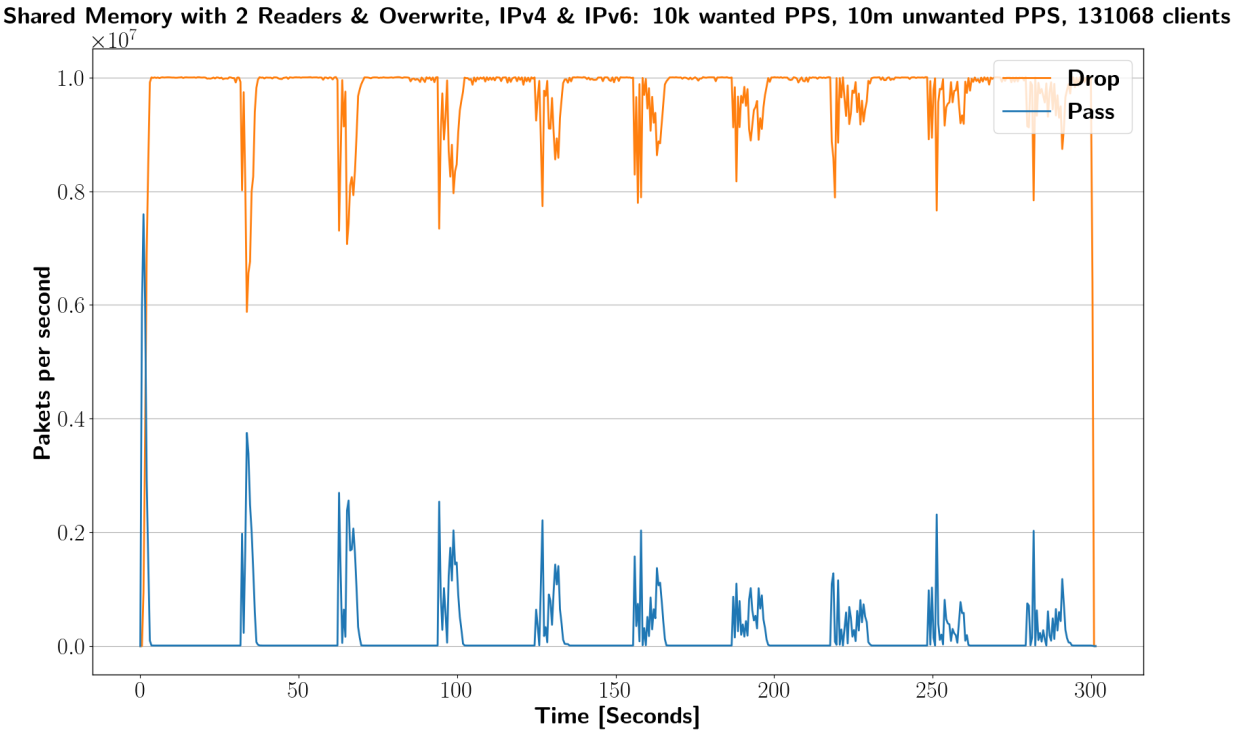
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
8095.28	8017.72	99.13	21	76.23

Figure 4.21: Some text



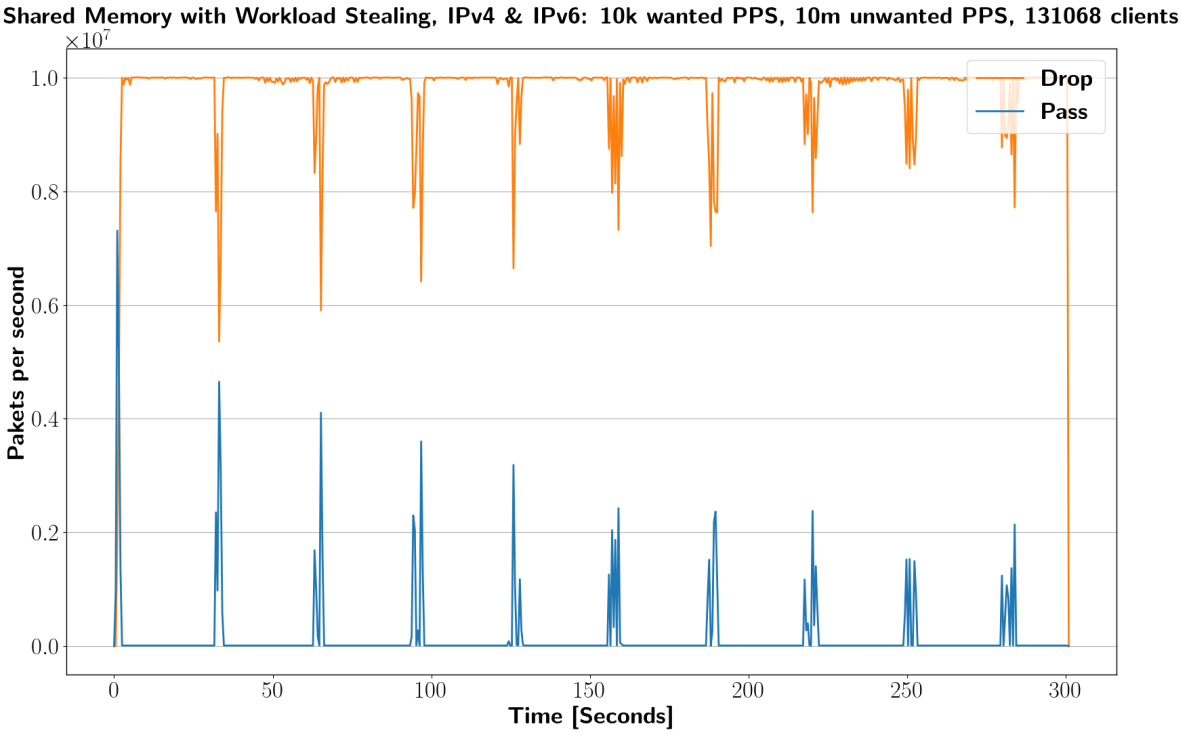
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2989.66	2906.36	97.44	17.67	77.19

Figure 4.22: Some text



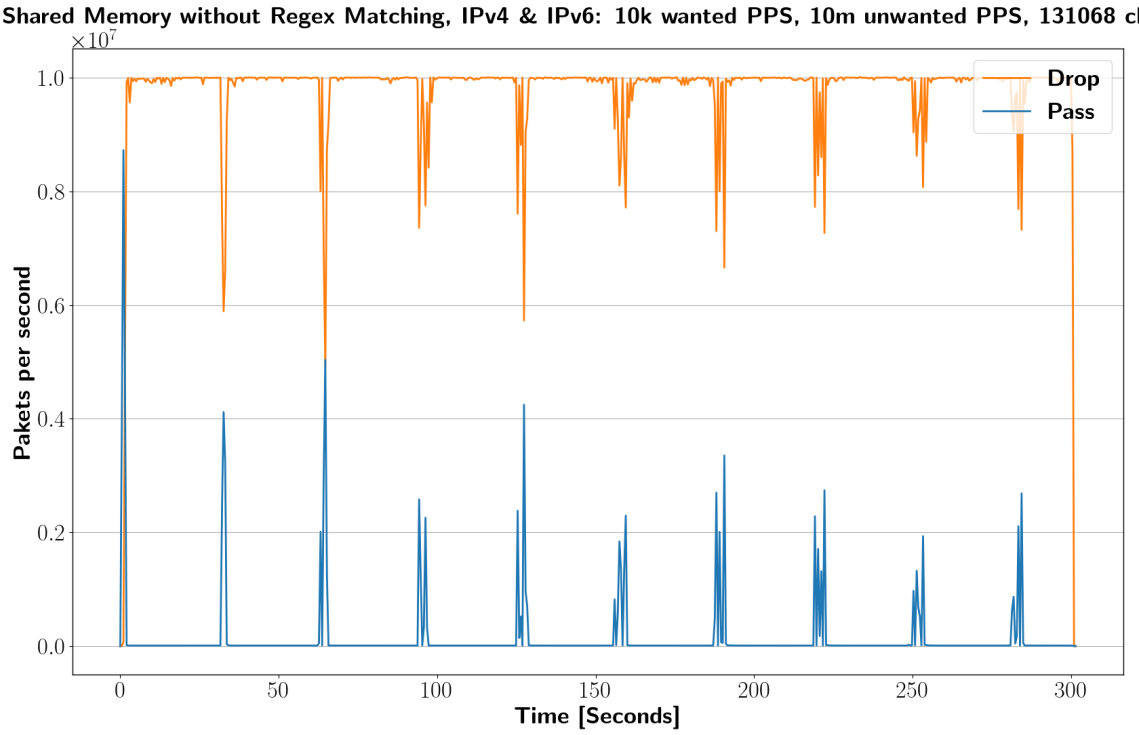
Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2988.9	2914.39	97.73	17.66	79.13

Figure 4.23: Some text



Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2991.23	2944.56	98.67	15.28	61

Figure 4.24: Some text



Total packets [10^6]	Packets dropped [10^6]	Relative drop [%]	Log messages [10^6]	CPU [seconds]
2989.56	2941.9	98.63	13.29	34.04

Figure 4.25: Some text

5 Conclusion

List of Figures

3.1 IPC Architecture 5
3.2 Shared Memory Architecture 6
3.3 Simplefail2ban Architecture 7

4.1 Fail2Ban Replication Measurements 1 14
4.2 IPC Architecture 15
4.3 Simplefail2ban, Logfile IPv4, 100k Packets per Second (PPS) 16
4.4 Simplefail2ban, Logfile IPv4, 1m PPS 17
4.5 Simplefail2ban, Logfile IPv4, 10m PPS 18
4.6 Simplefail2ban, Logfile IPv6, 100k PPS 19
4.7 Simplefail2ban, Logfile IPv6, 1m PPS 20
4.8 Simplefail2ban, Logfile IPv6, 10m PPS 21
4.9 Simplefail2ban, Logfile IPv4 & IPv6, 100k PPS 22
4.10 Simplefail2ban, Logfile IPv4 & IPv6, 1m PPS 23
4.11 Simplefail2ban, Logfile IPv4 & IPv6, 10m PPS 24
4.12 Simplefail2ban, Shared Memory, IPv4, 1m PPS 25
4.13 Simplefail2ban, Shared Memory, IPv4, 1m PPS 26
4.14 Simplefail2ban, Shared Memory, IPv4, 10m PPS 27
4.15 Simplefail2ban, Shared Memory, IPv6, 100k PPS 28
4.16 Simplefail2ban, Shared Memory, IPv6, 1m PPS 29
4.17 Simplefail2ban, Shared Memory, IPv6, 10m PPS 30
4.18 Simplefail2ban, Shared Memory, IPv4 & IPv6, 100k PPS 31
4.19 Simplefail2ban, Shared Memory, IPv4 & IPv6, 1m PPS 32
4.20 Simplefail2ban, Shared Memory, IPv4 & IPv6, 10m PPS 33
4.21 Simplefail2ban, Shared Memory, IPv4 & IPv6, 30m PPS 34
4.22 Simplefail2ban, Shared Memory 2 Readers 35
4.23 Simplefail2ban, Shared Memory 2 Readers with Overwrite 36
4.24 Simplefail2ban, Shared Memory with Workload Sharing 37
4.25 Simplefail2ban, Shared Memory without Regex Matching 38

List of Tables

3.1 Hash Collisions 8
3.2 IP String Conversion 8

4.1 Testbed Summary 12

List of Algorithms

3.1 Shared Memory Ringbuffer: Writer Parameters 4
3.2 Shared Memory Ringbuffer: Setup and Cleanup 8
3.3 Shared Memory Ringbuffer: Write API 8
3.4 Shared Memory Ringbuffer: Read API 9
3.5 Shared Memory Ringbuffer: Reader Parameters 9
3.6 Shared Memory Ringbuffer: Global Header 10
3.7 Shared Memory Ringbuffer: Segment Header Reader 10
3.8 Shared Memory Ringbuffer: Segment Header Writer 10
3.9 io_uring Getline 10
3.10 IP Hashtable 11

A Abbreviations

DUT	Device under Test
HIDS	Host-based Intrusion Detection System
IPC	Inter-Process Communication
IPS	Intrusion Prevention System
OS	Operating System
PPS	Packets per Second

B Source Files

For the sake of not having to chop down a forest to print this thesis, no full source files will be appended. The source code is available in a git repository at: https://gitup.uni-potsdam.de/raatschen/bachelorarbeit. Access can be requested through me, or the second supervisor Max Schrötter.

Bibliography

[1] Fail2Ban. Official Fail2Ban Website. https://www.fail2ban.org, 2011. Last visited on: 14.04.2023.

[2] James P Anderson. Computer security threat monitoring and surveillance. *Technical Report, James P. Anderson Company*, 1980.

[3] Dorothy E Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, pages 222–232, 1987.

[4] Giovanni Vigna and Christopher Kruegel. Host-based intrusion detection. In *Handbook of Information Security*, pages 701–713. California State University, 2006.

[5] Florian Mikolajczak. Implementation and Evaluation of an Intrusion Prevention System Leveraging eBPF on the Basis of Fail2Ban. https://www.cs.uni-potsdam.de/bs/teaching/docs/thesis/2022/mikolajczak.pdf, 2022. Master Thesis, Institute for Computer Science Operating Systems and Distributed Systems, University of Potsdam.